Chapter 8
I²C Module

This chapter describes the MCF5307 I²C module, including I²C protocol, clock synchronization, and the registers in the I²C programing model. It also provides extensive programming examples.

8.1 Overview

I²C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications requiring occasional communications over a short distance between many devices. The flexible I²C allows additional devices to be connected to the bus for expansion and system development.

The I²C system is a true multiple-master bus including arbitration and collision detection that prevents data corruption if multiple devices attempt to control the bus simultaneously. This feature supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

8.2 Interface Features

The I²C module has the following key features:

- Compatibility with I²C bus standard
- Support for 3.3-V tolerant devices
- Multiple-master operation
- Software-programmable for one of 64 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated START signal generation

- Acknowledge bit generation/detection
- Bus-busy detection

Figure 8-1 is a block diagram of the I^{2}C module.

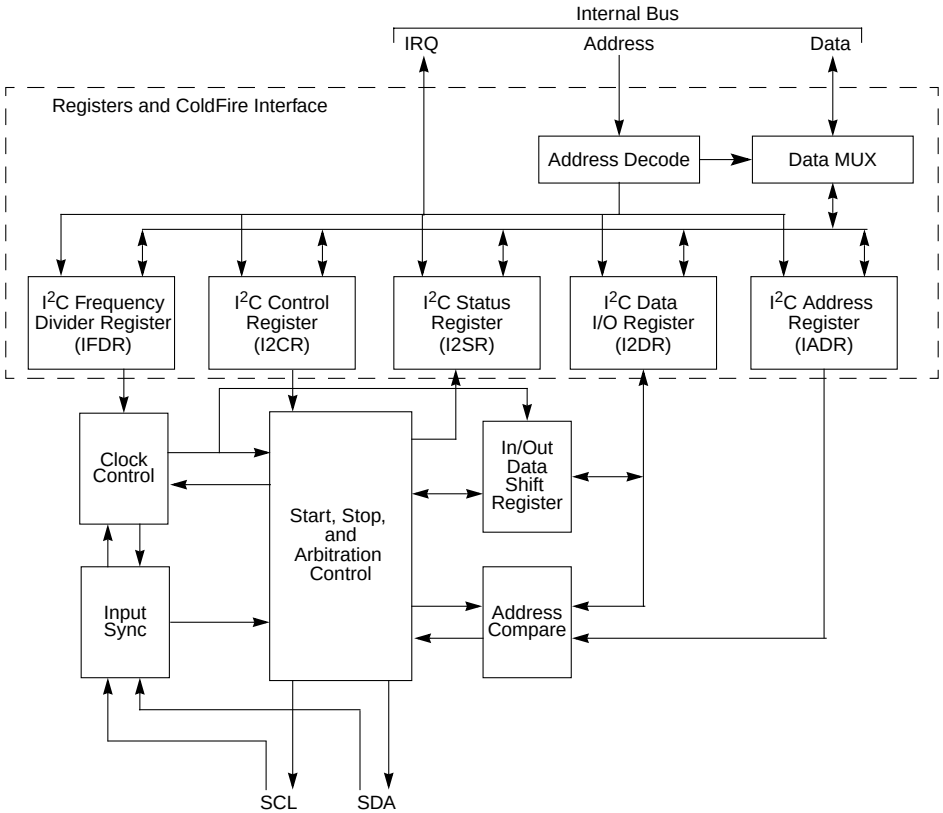

Figure 8-1. I^{2}C Module Block Diagram

Figure 8-1 shows the relationships of the I^{2}C registers, listed below:

- I^{2}C address register (IADR)
- I^{2}C frequency divider register (IFDR)
- I^{2}C control register (I2CR)
- I^{2}C status register (I2SR)
- I^{2}C data I/O register (I2DR)

These registers are described in Section 8.5, “Programming Model.”

8.3 I²C System Configuration

The I²C module uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. For I²C compliance, all devices connected to these two signals must have open drain or open collector outputs. (There is no such requirement for inputs.) The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I²C default is as slave receiver. Thus, when not programmed to be a master or responding to a slave transmit address, the I²C module should return to the default slave receiver state. See Section 8.6.1, “Initialization Sequence,” for exceptions.

NOTE:

The I²C module is designed to be compatible with the Philips I²C bus protocol. For information on system configuration, protocol, and restrictions, see *The I²C Bus Specification, Version 2.1*.

8.4 I²C Protocol

Normally, a standard communication is composed of the following parts:

1. START signal—When no other device is bus master (both SCL and SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in Figure 8-2). A START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

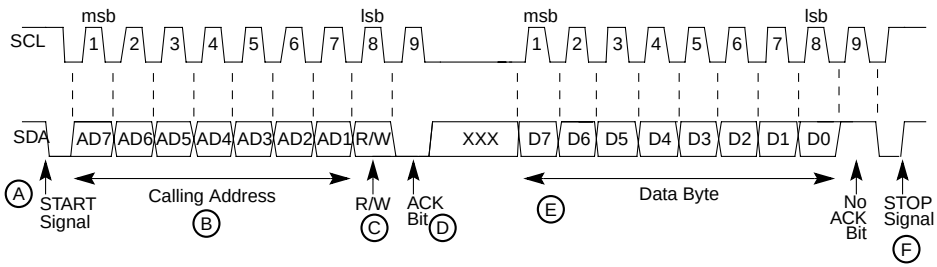

Figure 8-2. I²C Standard Communication Protocol

2. Slave address transmission—The master sends the slave address in the first byte after the START signal (B). After the seven-bit calling address, it sends the R/W bit (C), which tells the slave data transfer direction.

   Each slave must have a unique address. An I²C master must not transmit an address that is the same as its slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls SDA low at the ninth clock (D) to return an acknowledge bit.

3. Data transfer—When successful slave addressing is achieved, the data transfer can proceed (E) on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

   Data can be changed only while SCL is low and must be held stable while SCL is high, as Figure 8-2 shows. SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses.

   If it does not acknowledge the master, the slave receiver must leave SDA high. The master can then generate a STOP signal to abort the data transfer or generate a START signal (repeated start, shown in Figure 8-3) to start a new calling sequence.

   If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases SDA for the master to generate a STOP or START signal.

4. STOP signal—The master can terminate communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical high (F). Note that a master can generate a STOP even if the slave has made an acknowledgment, at which point the slave must release the bus.

Instead of signalling a STOP, the master can repeat the START signal, followed by a calling command, (A in Figure 8-3). A repeated START occurs when a START signal is generated without first generating a STOP signal to end the communication.

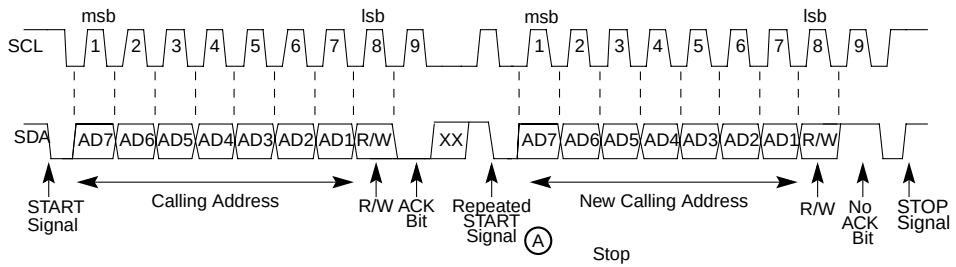

Figure 8-3. Repeated START

The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

8.4.1 Arbitration Procedure

If multiple devices simultaneously request the bus, the bus clock is determined by a synchronization procedure in which the low period equals the longest clock-low period among the devices and the high period equals the shortest. A data arbitration procedure

determines the relative priority of competing devices. A device loses arbitration if it sends logic high while another sends logic low; it immediately switches to slave-receive mode and stops driving SDA. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.

8.4.2 Clock Synchronization

Because wire-AND logic is used, a high-to-low transition on SCL affects devices connected to the bus. Devices start counting their low period when the master drives SCL low. When a device clock goes low, it holds SCL low until the clock high state is reached. However, the low-to-high change in this device clock may not change the state of SCL if another device clock is still in its low period. Therefore, the device with the longest low period holds the synchronized clock SCL low. Devices with shorter low periods enter a high wait state during this time (See Figure 8-4). When all devices involved have counted off their low period, the synchronized clock SCL is released and pulled high. There is then no difference between device clocks and the state of SCL, so all of the devices start counting their high periods. The first device to complete its high period pulls SCL low again.

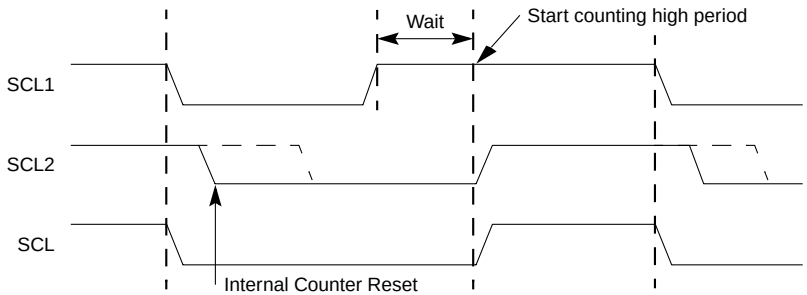

Figure 8-4. Synchronized Clock SCL

8.4.3 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfers. Slave devices can hold SCL low after completing one byte transfer (9 bits). In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases SCL.

8.4.4 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is longer than the master SCL low period, the resulting SCL bus signal low period is stretched.

8.5 Programming Model

Table 8-1 lists the configuration registers used in the I^{2}C interface.

Table 8-1. I^{2}C Interface Memory Map

MBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x280	I^{2}C address register (IADR) [p. 8-6]	Reserved		
0x284	I^{2}C frequency divider register (IFDR) [p. 8-7]	Reserved		
0x288	I^{2}C control register (I2CR) [p. 8-8]	Reserved		
0x28C	I^{2}C status register (I2SR) [p. 8-9]	Reserved		
0x290	I^{2}C data I/O register (I2DR) [p. 8-10]	Reserved		

NOTE:

External masters cannot access the MCF5307's on-chip memories or MBAR, but can access any I^{2}C module register.

8.5.1 I^{2}C Address Register (IADR)

The IADR holds the address the I^{2}C responds to when addressed as a slave. Note that it is not the address sent on the bus during the address transfer.

	7	6	5	4	3	2	1	0
Field	ADR							—
Reset	0000_0000							
R/W	Read/Write							
Address	MBAR + 0x280							

Figure 8-5. I^{2}C Address Register (IADR)

Table 8-2 describes IADR fields.

Table 8-2. I^{2}C Address Register Field Descriptions

Bits	Name	Description
7–1	ADR	Slave address. Contains the specific slave address to be used by the I^{2}C module. Slave mode is the default I^{2}C mode for an address match on the bus.
0	—	Reserved, should be cleared.

8.5.2 I²C Frequency Divider Register (IFDR)

The IFDR, Figure 8-6, provides a programmable prescaler to configure the clock for bit-rate selection.

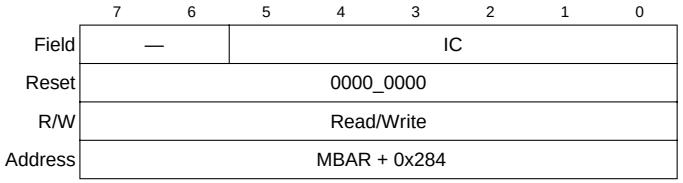

Figure 8-6. I²C Frequency Divider Register (IFDR)

Table 8-3 describes IFDR[IC].

Table 8-3. IFDR Field Descriptions

Bits	Name	Description
7–6	—	Reserved, should be cleared.
5–0	IC	I²C clock rate. Prescales the clock for bit-rate selection. Due to potentially slow SCL and SDA rise and fall times, bus signals are sampled at the prescaler frequency. The serial bit clock frequency is equal to BCLK0 divided by the divider shown below. Note that IC can be changed anywhere in a program.

IC	Divider	IC	Divider	IC	Divider	IC	Divider
0x00	28	0x10	288	0x20	20	0x30	160
0x01	30	0x11	320	0x21	22	0x31	192
0x02	34	0x12	384	0x22	24	0x32	224
0x03	40	0x13	480	0x23	26	0x33	256
0x04	44	0x14	576	0x24	28	0x34	320
0x05	48	0x15	640	0x25	32	0x35	384
0x06	56	0x16	768	0x26	36	0x36	448
0x07	68	0x17	960	0x27	40	0x37	512
0x08	80	0x18	1152	0x28	48	0x38	640
0x09	88	0x19	1280	0x29	56	0x39	768
0x0A	104	0x1A	1536	0x2A	64	0x3A	896
0x0B	128	0x1B	1920	0x2B	72	0x3B	1024
0x0C	144	0x1C	2304	0x2C	80	0x3C	1280
0x0D	160	0x1D	2560	0x2D	96	0x3D	1536
0x0E	192	0x1E	3072	0x2E	112	0x3E	1792
0x0F	240	0x1F	3840	0x2F	128	0x3F	2048

8.5.3 I²C Control Register (I2CR)

The I2CR is used to enable the I²C module and the I²C interrupt. It also contains bits that govern operation as a slave or a master.

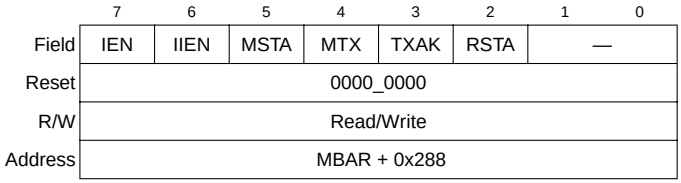

Figure 8-7. I²C Control Register (I2CR)

Table 8-4 describes I2CR fields.

Table 8-4. I2CR Field Descriptions

Bits	Name	Description
7	IEN	I²C enable. Controls the software reset of the entire I²C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next start condition is detected. Master mode is not aware that the bus is busy; so initiating a start cycle may corrupt the current bus cycle, ultimately causing either the current master or the I²C module to lose arbitration, after which bus operation returns to normal. 0 The module is disabled, but registers can still be accessed. 1 The I²C module is enabled. This bit must be set before any other I2CR bits have any effect.
6	IIEN	I²C interrupt enable. 0 I²C module interrupts are disabled, but currently pending interrupt condition are not cleared. 1 I²C module interrupts are enabled. An I²C interrupt occurs if I2SR[IIF] is also set.
5	MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4	MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When a slave is addressed, software should set MTX according to I2SR[SRW]. In master mode, MTX should be set according to the type of transfer required. Therefore, for address cycles, MTX is always 1.
3	TXAK	Transmit acknowledge enable. Specifies the value driven onto SDA during acknowledge cycles for both master and slave receivers. Note that writing TXAK applies only when the I²C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (that is, acknowledge bit = 1).
2	RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1–0	—	Reserved, should be cleared.

8.5.4 I²C Status Register (I2SR)

This I2SR contains bits that indicate transaction direction and status.

	7	6	5	4	3	2	1	0
Field	ICF	IAAS	IBB	IAL	—	SRW	IIF	RXAK
Reset	1000_0001							
R/W	R			R/W	R		R/W	R
Address	MBAR + 0x28C							

Figure 8-8. I²CR Status Register (I2SR)

Table 8-5 describes I2SR fields.

Table 8-5. I2SR Field Descriptions

Bits	Name	Description
7	ICF	Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by the falling edge of the ninth clock of a byte transfer.
6	IAAS	I²C addressed as a slave bit. The CPU is interrupted if I2CR[IIEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CR clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5	IBB	I²C bus busy bit. Indicates the status of the bus. 0 Bus is idle. If a STOP signal is detected, IBB is cleared. 1 Bus is busy. When START is detected, IBB is set.
4	IAL	Arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.) • SDA sampled low when the master drives high during an address or data-transmit cycle. • SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it.
3	—	Reserved, should be cleared.
2	SRW	Slave read/write. When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I²C module is a slave and has an address match. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1	IIF	I²C interrupt. Must be cleared by software by writing a zero to it in the interrupt routine. 0 No I²C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1). Set when one of the following occurs: • Complete one byte transfer (set at the falling edge of the ninth clock) • Reception of a calling address that matches its own specific address in slave-receive mode • Arbitration lost
0	RXAK	Received acknowledge. The value of SDA during the acknowledge bit of a bus cycle. 0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus 1 No acknowledge signal was detected at the ninth clock.

8.5.5 I²C Data I/O Register (I2DR)

In master-receive mode, reading the I2DR, Figure 8-9, allows a read to occur and initiates next byte data receiving. In slave mode, the same function is available after it is addressed.

	7	6	5	4	3	2	1	0
Field	D							
Reset	0000_0000							
R/W	Read/Write							
Address	MBAR + 0x290							

Figure 8-9. I²C Data I/O Register (I2DR)

8.6 I²C Programming Examples

The following examples show programming for initialization, signalling START, post-transfer software response, signalling STOP, and generating a repeated START.

8.6.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized, as follows:

1. Set IFDR[IC] to obtain SCL frequency from the system bus clock. See Section 8.5.2, "I²C Frequency Divider Register (IFDR)."
2. Update the IADR to define its slave address.
3. Set I2CR[IEN] to enable the I²C bus interface system.
4. Modify the I2CR to select master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE:

If IBSR[IBB] when the I²C bus module is enabled, execute the following code sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were just power-cycled on.

```
I2CR = 0x0
I2CR = 0xA
dummy read of I2DR
IBSR = 0x0
I2CR = 0x0
```

8.6.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. On a multiple-master bus system, IBSR[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB = 0), the START signal

and the first byte (the slave address) can be sent. The data written to the data register comprises the address of the desired slave and the lsb indicates the transfer direction.

The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I²C is busy after writing the calling address to the I2DR before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

```
CHFLAG  MOVE.B I2SR,-(A0);Check I2SR[MBB]
        BTST.B #5, (A0)+
        BNE.S CHFLAG;If I2SR[MBB] = 1, wait until it is clear
TXSTART MOVE.B I2CR,-(A0);Set transmit mode
        BSET.B #4,(A0)
        MOVE.B (A0)+, I2CR
        MOVE.B I2CR, -(A0);Set master mode
        BSET.B #5, (A0);Generate START condition
        MOVE.B (A0)+, I2CR
        MOVE.B CALLING,-(A0);Transmit the calling address, D0=R/W
        MOVE.B (A0)+, I2DR
IFREE   MOVE.B I2SR,-(A0);Check I2SR[MBB]
            ;If it is clear, wait until it is set.
        BTST.B #5, (A0)+;
        BEQ.S IFREE;
```

8.6.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IIEN]. Software must first clear IIF in the interrupt routine. ICF is cleared either by reading from I2DR in receive mode or by writing to I2DR in transmit mode.

Software can service the I²C I/O in the main program by monitoring IIF if the interrupt function is disabled. Polling should monitor IIF rather than ICF because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; that is, the address is sent. If master receive mode is required (I2DR[R/W], I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see Figure 8-10).

```
I2SR    LEA.L I2SR,-(A7);Load effective address
        BCLR.B #1,(A7)+;Clear the IIF flag
        MOVE.B I2CR,-(A7);Push the address on stack,
```

```
        BTST.B #5,(A7)+;check the MSTA flag
        BEQ.S SLAVE;Branch if slave mode
        MOVE.B I2CR,-(A7);Push the address on stack
        BTST.B #4,(A7)+;check the mode flag
        BEQ.S RECEIVE;Branch if in receive mode
        MOVE.B I2SR,-(A7);Push the address on stack,
        BTST.B #0,(A7)+;check ACK from receiver
        BNE.B END;If no ACK, end of transmission
TRANSMITMOVE.B DATABUF,-(A7);Stack data byte
        MOVE.B (A7)+, I2DR;Transmit next byte of data
```

8.6.4 Generation of STOP

A data transfer ends when the master signals a STOP, which can occur after all data is sent, as in the following example.

```
MASTX   MOVE.B I2SR, -(A7);If no ACK, branch to end
        BTST.B #0,(A7)+
        BNE.B END
        MOVE.B TXCNT,D0;Get value from the transmitting counter
        BEQ.S END;If no more data, branch to end
        MOVE.B DATABUF,-(A7);Transmit next byte of data
        MOVE.B (A7)+,I2DR
        MOVE.B TXCNT,D0;Decrease the TXCNT
        SUBQ.L #1,D0
        MOVE.B D0,TXCNT
        BRA.S EMASTX;Exit
END     LEA.L I2CR,-(A7);Generate a STOP condition
        BCLR.B #5,(A7)+
        EMASTX RTE;Return from interrupt
```

For a master receiver to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last data byte. This is done by setting I2CR[TXAK] before reading the next-to-last byte. Before the last byte is read, a STOP signal must be generated, as in the following example.

```
MASR    MOVE.B RXCNT,D0;Decrease RXCNT
        SUBQ.L #1,D0
        MOVE.B D0,RXCNT
        BEQ.S ENMASR;Last byte to be read
        MOVE.B RXCNT,D1;Check second-to-last byte to be read
        EXTB.L D1
        SUBI.L #1,D1;
        BNE.S NXMAR;Not last one or second last
        LAMAR BSET.B #3,I2CR;Disable ACK
        BRA NXMAR

ENMASR  BCLR.B #5,I2CR;Last one, generate STOP signal
NXMAR   MOVE.B I2DR,RXBUF;Read data and store RTE
```

8.6.5 Generation of Repeated START

After the data transfer, if the master still wants the bus, it can signal another START followed by another slave address without signalling a STOP, as in the following example.

```
RESTART MOVE.B I2CR,-(A7);Repeat START (RESTART)
        BSET.B #2, (A7)
        MOVE.B (A7)+, I2CR
        MOVE.B CALLING,-(A7);Transmit the calling address, D0=R/W-
        MOVE.B CALLING,-(A7);
        MOVE.B (A7)+, I2DR
```

8.6.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. If IAAS is set, software should set the transmit/receive mode select bit (I2CR[MTX]) according to the I2SR[SRW]. Writing to the I2CR clears the IAAS automatically. The only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer can now be initiated by writing information to I2DR for slave transmits, or read from I2DR in slave-receive mode. A dummy read of I2DR in slave/receive mode releases SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[RXAK] must be tested before sending the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which software must switch it from transmitter to receiver mode. Reading I2DR then releases SCL so that the master can generate a STOP signal.

8.6.7 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes master. Hardware immediately switches devices that lose arbitration to slave receive mode. Data output to SDA stops, but SCL is still generated until the end of the byte during which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR[IAL] = 1 and I2CR[MSTA] = 0.

If a device that is not a master tries to transmit or do a START, hardware inhibits the transmission, clears MSTA without signalling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed attempt to engage the bus. When considering these cases, the slave service routine should first test IAL and software should clear it if it is set.

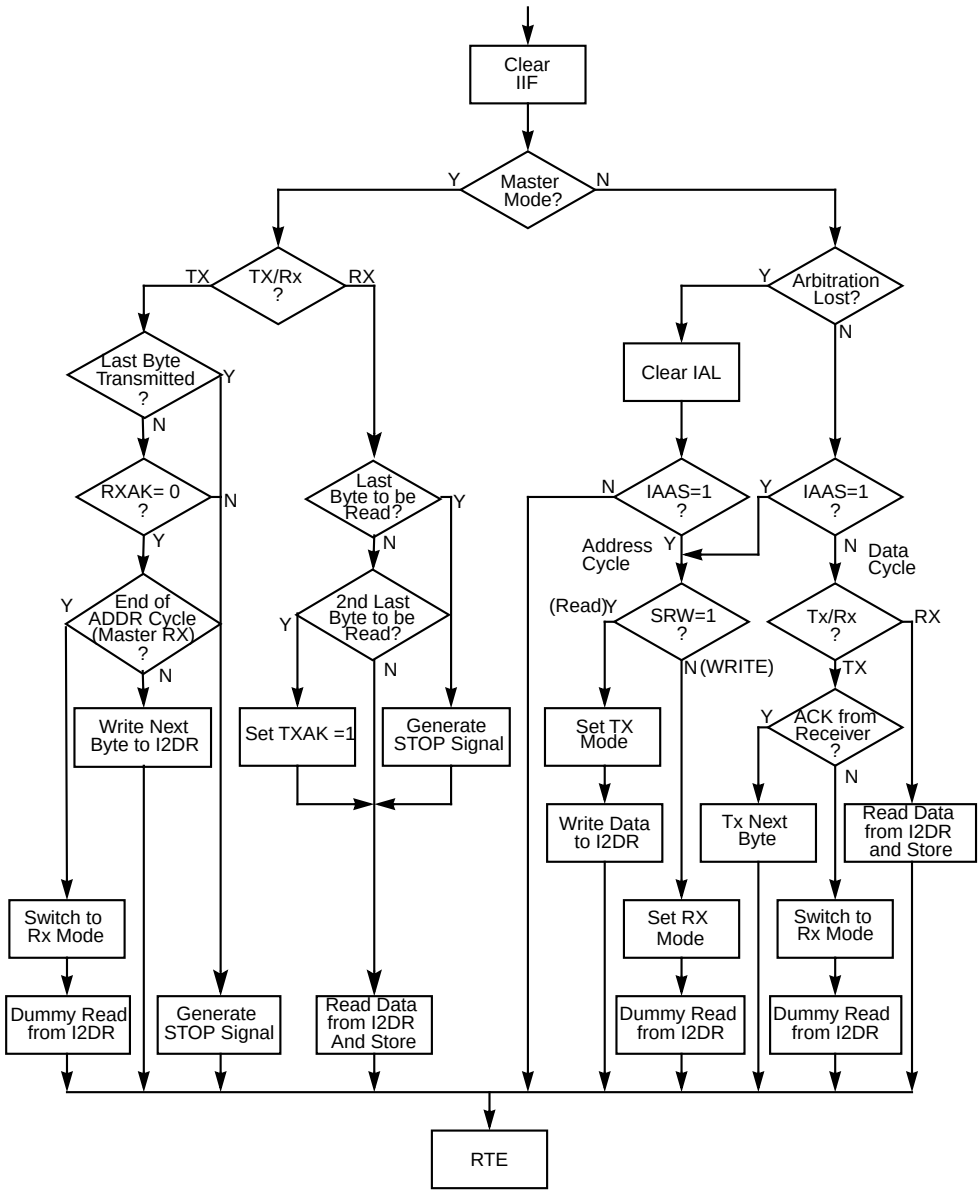

Figure 8-10. Flow-Chart of Typical I²C Interrupt Routine